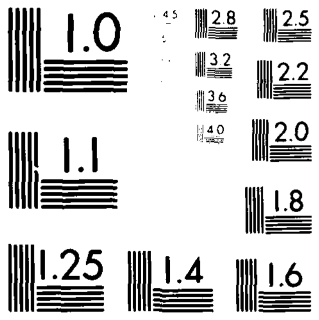


MICROCOPY RESOLUTION TEST CHART

NATIONAL BUREAU OF STANDARDS 1963 A

LEVEL II (12)

ADA085894

Problem Solving in a Natural Language Environment

Working Paper 22

by

Bradley Alan Goodman

Coordinated Science Laboratory
University of Illinois at Urbana/Champaign
Urbana, IL 61801

21 July 1979

Keywords and phrases: natural language understanding, problem solving, vague and complex requests, frames, relational data bases, natural language front ends, JETS, PLANES.

Abstract

The kinds of requests that can be currently handled by natural language data base systems are constrained mainly to simple queries to retrieve information from the data base. The requests must be completely specified by the user (though certain information can be assumed from past context). This paper is a proposal for a Ph.D. thesis that explores requests of a more complicated nature. The goal is to take vague and complex requests from users and turn them into well-defined problems. Missing information will be filled in through world knowledge or from the current dialog context. The transformation of the request into a well-defined problem and the generation of a plan to solve the problem will be guided by a set of problem solving frames.

This work was supported by the Office of Naval Research under Contract N00014-75-C0612.

80 5 5 076

Problem Solving in a Natural Language Environment

by

Bradley Alan Goodman

B.S., Carnegie-Mellon University, 1975
M.S., University of Illinois, 1978

Thesis Proposal

University of Illinois at Urbana-Champaign, 1979

Thesis Advisor: Prof. David L. Waltz

Urbana, Illinois
July 21, 1979

July 21 1979

TABLE OF CONTENTS

I Introduction 1

I.1 What is a problem? .1

I.2 What is a solution? .1

I.3 Finding the correct representation. 1

I.4 I propose to... 2

I.4.a General and Problem Solving Frames 3

I.4.b Heuristics .3

II Representations for Knowledge 5

II.1 Frames 5

II.2 Knowledge Representation Systems 6

III Weaknesses of PLANES 7

IV Our Goals in JETS 10

IV.1 Problem Solving Frames 12

IV.2 Well-Defined Problems 12

IV.3 Purpose of Problem Solving Frames 13

IV.4 A Scenario 13

IV.5 Handling Vague and Complex Requests 17

IV.5.a How can the system attempt this problem? 18

V The Approach 22

V.1 Hierarchy of Frames 22

V.2 Solving the Problems 23

V.3 Handling Incomplete Problems 24

V.4 Assimilation 30

V.5 Efficiency 30

VI Conclusion 32

APPENDICES 33

A Historical Look at Previous Representations for Problems 33

A.1 State Spaces and Problem Reduction Operators 33

A.2 Generate and Test Paradigm 33

A.3 Predicate Calculus 35

B Other Approaches to Problem Solving 37

B.1 Expert Problem Solvers 37

B.2 PSI: A Program Synthesis System 38

C Knowledge Representation Systems 40

C.1 FRL 40

C.2 KRL 41

C.3 KLONE 42

VII References 46

LIST OF FIGURES

FIGURE Page

1. An Interpretation Rule for Sets 11
2. A Scenario 14
3. COMPARE Frame 15
4. Defining Units 16
5. An Overview of JETS 21
6. A Simple Example of Problem Solving 25
7. Generate and Test Paradigm 34
8. Overview of PSI 39
9. Sample Unit of KRL[BOBROW, p.g. 10] 43
10. KLONE Structures[BRACHM78, p.g. 33] 45

I

Introduction

I.1 What is a problem?

The answer to "what is a problem?" is, in fact, not at all clear. Webster's dictionary defines it as "a question raised for inquiry, consideration, or solution" [WEBSTER]. A problem certainly is related to the desires of someone to achieve some goal or to obtain certain information. It is actually easier to define what it means to have a problem than what a problem is. Having a problem means to "search consciously for some action appropriate to a clearly conceived, but not immediately attainable, aim"[POLYA62]. In AI, problems have often been defined as an initial state and a goal state. Optionally associated with these states are a set of operators, a set of constraints to follow, etc.

I.2 What is a solution?

A solution is a process for solving a problem. It is a plan of action that explains in step-by-step format the course that must be followed to find the answer to a problem. For the most part there is no agreed upon set of methods for solving problems. A problem is considered solved when the solution generated can be "plugged" back into the problem itself to show that it works. One interesting thing about solutions is that very often they are the best way for describing a problem.

I.3 Finding the correct representation.

A representation is a data type used to define an analogous problem that we need to solve in order to solve the real problem.(*) It is a notational device that permits us to express the real problem in a format more conducive to problem solving. In fact, the better the representation, the easier it normally is to solve the problem. The best representations normally have features that correspond closely to those of the real problem so that it is easy to translate the solution.

The representations chosen are constrained to those that are reasonable to implement on a computer. This fact should be kept in mind when developing

* In the case of theorem proving, no analogy is needed.

a representation so that the majority of time spent during problem solving is not wasted on massaging the data representation. This issue of the relationship between different ways of formulating a problem and the efficiency with which the computer can be expected to find a solution is a prerequisite to the design of procedures to automatically construct an appropriate representation for a problem.

I.4 I propose to...

One of the central issues in a natural language question answering environment is how to convert vague and complex requests into an internal representation that can be characterized as a well-defined problem. Well-defined means that the problem statement itself, when coupled with world knowledge (from the data base and past context), is sufficient to permit a solution.

This problem has not been tackled in previous question answering or general problem solving systems. The closest attempt at handling complex (but not vague) problems would be by the expert problem solvers that are designed to work in one environment. These systems contain a fairly complete knowledge base of important facts in a domain. Existing systems have expertise in organic chemistry (DENDRAL [FEIGENBA]), medicine (MYCIN [SHORTLIF]) (INTERNIST [POPLE]), and knowledge base construction (TEIRESIAS [DAVIS76]). They are generally menu-driven (i.e. multiple choice questions are asked of the user), making it easier to force the user to specify a problem completely. A large data base of production rules is used to encode specific knowledge about the domain. The systems typically work by initiating a dialogue (via a set of multiple choice questions) requesting information for their short term memory until production rules begin firing, giving suggestions or requesting more specific information.

My proposal is to go significantly beyond the work done with expert problem solvers in the following ways:
(1) My system will allow a user to communicate his or her problem in English to the system instead of via a sequence of answers to multiple choice questions.
(2) It will not limit itself to one very specific domain. For example, DENDRAL's principal problem solving ability involves deciding what the chemical structure of an organic molecule is when given a mass spectrogram and chemical formula of the molecule as input. While the task of identifying a chemical structure is very involved, the statement of the problem itself is limited to a very narrow channel (the spectrograms and formula).

TEIRESIAS [DAVIS76] is not quite as limited; it allows an expert designer to enter through a dialogue the production rules which encode TEIRESIAS' knowledge. Davis also encoded production rules that know general things about other production rules and cause those rules to be updated, depending on changes to the knowledge base.

(3) My system will handle vague as well as complex questions. Vague questions

are incompletely specified, i.e. certain relevant information is missing. There may also be irrelevant information that must be detected and ignored. _Complex questions_ are ones that go beyond a simple request for direct retrieval of information. They can be long descriptions of a task to be performed rather than single sentences. They may require higher level processing (e.g. analysis) than can be specified in a formal query.

I.4.a _General and Problem Solving Frames_

Problem solving frames are to be used to encode some of the kinds of common-sense problem solving skills people have. People know many ways to solve simple problems and have the ability to ignore irrelevancies to get to the heart of a complex problem. Continued thinking about a problem can result in a narrowing (and, every so often, expanding) of the concepts thought to be important in a problem. Eventually a group of relevant concepts will result. In essense, these relevancy judgements result in the focussing of one's attention to the essential features of a problem--resulting in an increase in efficiency.

The _general knowledge_ frames in my proposed system will encode small "chunks" of information about concepts that people can use when trying to focus on the important facets of a problem. They will store information about concepts, events, information in the data base, etc. (the detailed data base information concerning relations, field names, etc. will be stored in a model of the data base). These chunks allow people to ignore details that will most likely be useful only when actually executing a plan to find an answer; they make it easier to judge what is relevant in a problem statement. They also make it easier to choose a methodology for solving the problem because fewer details will have to be checked when deciding whether or not a particular solution technique applies.

I.4.b _Heuristics_

Heuristics appear to encode a fair amount of the problem solving skills people use. One important group of heuristics are those that will be used (in conjunction with the general knowledge frames) to help choose what aspects of a problem statement are important (and at the same time, filtering out irrelevant information). I feel the following heuristics are beneficial and should be employed in my system:

1 Initially assume each English sentence defines _one_ particular (mutually-exclusive) aspect of the problem. This is used to help begin the search for relevant problem solving frames. Once a frame is activated, it can go beyond the particular sentence that activated it to look for more pertinent information (it can look not only at the other sentences in the user's request but it can examine the "everyday" knowledge stored in the general knowledge frames).

2 In each sentence, look for _relevant_ information [HAYES]:

(a) look for phrases describing time,
(b) try to identify when groups of objects (sets) are specified,
(c) give questions higher priority than other input, since they are likely to be highly relevant,
(d) look for phrases describing concepts, events, etc. that have been recently mentioned (i.e. use past CONTEXT to aid in understanding the request),
(e) assign a "weight" to each piece of information found (both relevant and irrelevant),
(f) process the concepts that seem most relevant first;

3 for each relevant concept found, find the general knowledge frame that describes the concept, and fill it in with the particular details; having a general description will make it easier to make inferences.

When trying to find problem solving frames that can be used to help solve the problem, the system will: (1) examine the concepts marked relevant according to the way they were weighted, (2) select the best sequence of candidate problem solving frames, and (3) order the problem solving frames.

The detection of an incompletely-specified problem is an important task in problem solving. The set of problem solving and general knowledge frames selected above will be examined (in order) to see if any information tagged important is missing from the problem statement. Since the level of description of a concept in the frames may differ from what is in the problem statement, the system must take the most specific of the two descriptions and generalize it (using information in the general knowledge hierarchy). Once generalized, it should be possible to tell if the two concepts are similar. Once this is known, then more specific information found in the frames can be added into the problem statement to make it complete (the user will be informed, and asked if, the information added is correct).

II

Representations for Knowledge

There is a need for a new representation for problems and their solutions for use in a natural language environment. Problem solving, like language understanding, involves organizing a large memory. It is desirable that any representation structure selected be more "natural" (i.e. easier to translate to and from English) than those of the predicate calculus formalism and be similar in design to those used by the rest of a natural language system to store the general knowledge about the domain of discourse. This is particularly important because the problem solving mechanism will need to interact closely with the general knowledge structures to apply its expertise to a particular domain.

II.1 Frames

A frame [MINSKY75] [TENNAN80]is a specialized data structure that can represent some concept (i.e. knowledge about a very limited domain). Languages for writing frame systems usually have associated with them a set of functions for defining frames and for storing and retrieving values from them. A frame is composed of a group of slots where each slot defines one aspect of the concept represented by the frame. These slots can be assigned particular values or can have default values assigned to them. Procedures can also be associated with a slot and can become activated depending on the values stored in the slot. Slots can also point to other frames where a deeper explanation can be found for that particular aspect of the concept represented by the frame. Frames are connected to other frames through the use of links. These links set up a hierarchy of frames, one of which is based on SUBCONCEPT/SUPERCONCEPT relationships. One can move up and down these links to pass information--with more specific frames "inheriting" information from an ancestor.

Before one can use frames to portray some thought or concept, must first try to find a particular frame in the frame system that encompasses most of what one wishes to represent. One can then form a specific "instance" of that frame using the particular details of the thought or concept. When this has been accomplished, the frame is said to have been "instantiated". The instantiation of a frame could be deactivated at any time should the topic under investigation change or should a better match be found when more details become available. The next step requires searching for properties not in the frame through the inheritance mechanism (or via frame transformation, analogy links, context-relatedness links, etc). A couple of capabilities provided in some frame systems, that demonstrate the use of this mechanism, include the ability to compare two frames to each other and to view the concept represented by one frame as that represented by another. To compare two frames to each other, deductive rules can be applied by the frame system.

Such a set of general rules might check if the two concepts under comparison have common ancestors under subconcept/superconcept links. Frames can also be used to view one thing as another. The procedure to view one concept as another must first try to generate a mapping between the aspects represented by the slots of the frame depicting the first concept to those of the second. When this is achieved, the mapping can be used to generate another mapping that can map the "values" of those slots onto each other.

II.2 Knowledge Representation Systems

Several general knowledge representation schemes have been proposed over the last several years. FRL (Frame Representation Language) [ROBERTS], KRL (Knowledge Representation Language) [BOBROW] and KLONE [BRACHM78] are three of the major representations. FRL is by far the most primitive--providing the user with a set of procedures to build up a hierarchy of frames but leaving the details of the structure itself to the user. KRL is an attempt to unify procedural knowledge with declarative forms. KRL claims to be both a high level AI programming language and a theory of knowledge representation. KLONE is built around a semantic network structure. The network, however, employs a fixed set of primitive links instead of links that can be defined to be anything. This makes any relationship shown in the network easier to understand. KLONE also provides means for the properties of one node to be inherited by others. (*)

* See Appendix C for descriptions of these knowledge representation systems.

III

Weaknesses of PLANES

The PLANES system was designed to accept questions in English from casual users about aircraft maintenance and flight data in a subset of the U.S. Navy 3-M (Maintenance and Material Management) data base. The system was designed to handle complex syntactic structures, abbreviations, pronoun reference and ellipsis. It tried to give back explicit answers and not just retrieve a file. Minor errors--such as spelling and grammatical errors--were tolerated.

Parsing of inputs in PLANES involved four operations: (1) putting all words and phrases in canonical form and correcting spelling; (2) matching against prestored phrase patterns (ATN subnets) and setting a context register (history keeper); (3) matching the context register values against concept case frame patterns; and (4) filling in missing contextual information needed to form a meaningful query. The most difficult but valuable task was the use of the concept case frames. Concept case frames enumerate the kinds of questions understood by the system. They consist of an act (which is related to a verb) and a list of noun phrases (referred to by subnet/context register names) which can occur in a meaningful way with the act. As an example, concept case frames synonymous with require might be (*require *planetype *maintenance-type) and (*require *part *maintenance-type). Together with the ATN subnets, concept case frames form a semantic grammar [HALLIDAY].

When constituents of a sentence are missing (as in ellipsis) or replaced by pronouns or referential phrases, the system can suggest what type of phrase is necessary to complete the concept by finding all concept case frames which match the rest of the sentence. If only one concept case frame matches, we are done; if more than one matches, then the reference to which constituents were present in the previous sentence is usually adequate to decide which one to use. If more than one concept case frame remains, the user is asked which referent is correct.

The query generator [GOODMA77] allows the system to construct formal queries directly from the semantic constituents of a request--even when no concept case frame matches the request. The query generator must: (1) decide which relations (files and card types) to look at to retrieve the information necessary for answering the user's request; (2) decide what domains (data fields) to return from the relations which are searched; (3) decide what semantic constituents should be represented as predicates; (4) decide what operations should be performed on the fields returned; (5) translate field values into internal data base codes; and (6) decide how to arrange the output data. The query generator utilizes heuristics to analyze the semantic constituents in order to generate the information mentioned above to construct a formal query. It is capable of handling most of the simple requests made of PLANES.

Numerous weaknesses with PLANES stood out after tests with casual users were performed in early 1978 [TENNAN79]. PLANES could handle only simple requests--single sentence inputs rather than long descriptions of the task to be performed. No complex requests--beyond the extraction of value(s) from field(s) over a set of constraints and the application of unsophisticated functions to those fields--could be analyzed. The complexity of the kinds of questions answerable by PLANES can be seen below.

```
how many A7s flew > 10 flights in 1973?
in 1972?
how many did they fly in 1974?
what maintenances were performed on it between
                            May 16 and 17 1969?
for planes 3 and 5, how many catapult flights
      were flown between jan 1 and jun 30 74?
how man" flights and flight hours were flown by
                              plane 3 in jan 73?
```

The responses to inquiries to the system could only be in one of the following forms: (1) the value(s) of a field found during the data base search; (2) the result of applying simple functions such as SUM, MIN and MAX to the values; (3) a table showing the values for several different fields; and (4) a histogram showing the corresponding values of two fields. This means that no analysis of the retrieved data is performed by PLANES. All the processing of the data had to be done in the query langauage itself (and in _one_ query only) instead of leaving more complicated processing to experts. Below you will find some requests that are relatively simple, but, nevertheless, difficult to _ever_ answer in PLANES.

```
Find the maintenances that take the longest times.
Which maintenances occur most often?
Compare the flights flown by ship-based planes as
   opposed to land-based planes during June 1972.
Were the number of maintenances performed to those
                                planes very large?
```

PLANES lacked a "good" internal representation of the problem asked by the user. A filled-in concept case frame was the only representation of the semantic interpretation of the request. Thus it provided an unordered list of the constituents of the initial request classified by semantic categories, along with some "hints" on how to generate a formal query to answer the request. The problem solving frames that I describe later actually stem from the way I originally envisioned concept case frames. They originally were to provide detailed knowledge about the domain of discourse, the structure of the data base, and ways of solving problems. Thus, my actual implementation for PLANES ended up being far short of my original conception.

Another problem with PLANES was that there was no easy way to try to recover from misunderstood requests because no "partial" understanding was possible. The system could only take the phrases that were parsed, and, using heuristics, "guess" what the user really asked. These heuristics used no world knowledge but simply relied on general facts such as a field name parsed with a particular value could be used as a predicate in the formal query and ones parsed without a particular value are probably fields whose values are to be returned. The addition of knowledge about the domain would have made it possible to make more intelligent guesses about what the user wanted and could have at least made it possible to ask the user intelligently to fill in particular gaps.

All of the above mentioned criticisms for PLANES can most certainly apply to other natural language data base systems that exist today. (*) I hope to attack this set of general, pervasive problems and not just those inherent in PLANES.

* I have personally had contact with the RENDEZVOUS system [CODD] and have used the parser for the LUNAR system [WOODS72]. An attempt to solve some of these problems is being attempted by the RENDEZVOUS group [HEMPHILL].

IV

<u>Our Goals in JETS</u>

The evaluation of the PLANES system [TENNAN79] led to the realization that the performance of a question answering system can be divided into two parts. The first part, the conceptual coverage of a system, is the set of concepts the system can handle. The second part, the linguistic coverage of a system, is the set of linguistic features that have been incorporated to permit variations in the way concepts are referenced. Linguistic converage encompasses such things as syntactic structure, anaphoric reference and ellipsis. Whether or not the conceptual and linguistic coverage is adequately represented in the system can be discovered by monitoring the handling of user's requests to the system. These measures have been called conceptual and linguistic completeness, respectively [TENNAN79].

To make it feasible to provide a wide conceptual coverage of the data base and its contents, of the events the data describes, and of the interaction between JETS and a user, a large store of knowledge must be created. We have decided to organize the knowledge of JETS into a network of frames. The frame will be our basic unit of representation. Most individual frames will be linked into the network using generality/specificity links. This will put the most general concepts higher up in the tree of frames and information can then be inherited down the tree. One nice feature of such a hierarchy is that it promotes the use of the most general concepts first when trying to interpret an input.

We have decided to implement the conceptual component of JETS in FRL--the language for building frame-based systems mentioned earlier. FRL was chosen because it was implemented at a level primitive enough to allow us much flexibility in defining our own set of knowledge structures.(*) This was particularly relevant because the implementation of JETS requires representing so many different kinds of knowledge--knowledge about the data base, the domain, the user environment (context, etc.), the query language, etc.

The use of frames in JETS will be done primarily by a set of interpretation rules. These rules will be implemented as production rules. Each rule will have a pattern or condition to match and an associated action to be performed. Implementing the interpretation rules as a production system will make it easier to insert and delete rules. Thus the range of the semantic interpreter can be expanded and contracted without having to understand or change the other rules. An example of an interpretation rule for sets can be seen below in figure 1 [FININ79].

* FRL is so flexible that it should be possible to implement both KRL and KLONE using it.

```
if <concept> modifies a <set> then:

        find the typical member of the <set>.

        find an applicable rule which interprets the
          modification of typical member by <concept>.

        invoke the rule on the typical member.

        invoke the rule on each of the members of the
         <set>.

        return the newly modified <set>.
```

An Interpretation Rule for Sets

figure 1

IV.1 Problem Solving Frames

The description of JETS so far has centered around ways of expanding the coverage of the system. However, nothing has been done to help the system develop a plan to extract the required information from the data base in the form the user intended. This is where problem solving frames are introduced. Problem solving frames are to be used to describe problem domains and search strategies. They can range from very general sketches of a particular series of problem environments to specific suggestions on how to answer a certain request. The latter problem solving frame might even be encoded in English where a sequence of English instructions are given on how to put all of the information together at the end to answer the main request. Problem solving frames will have to work jointly with the frames used during the parsing and semantic analysis stages. Those frames describe the objects and events of the JETS' environment and how they are related to the data in the data base. Information gathered by those frames can be used to extract appropriate values to fill the empty slots in the problem solving frames in order that the whole frame may become instantiated.

IV.2 Well-Defined Problems

Before describing the details of problem solving frames, a description of what a "problem" is should be discussed. A well-defined problem has been defined in the literature [NEWELL72] to satisfy the following criteria:

1 the problem is expressed in terms the solver can understand,

2 all information necessary for solving the problem should be in the problem statement itself and in the world knowledge available to the problem solver,

3 the form of the solution is exactly specified, and

4 there must be a systematic (algorithmic) way(s) for testing a proposed solution.

In our natural language query system environment (in particular with our experience with PLANES) we find that many requests by users are not normally well-defined, i.e. they do not satisfy the criteria above. Before such requests can be handled, it is necessary to make the request a well-defined problem. Previous systems have brought in this additional knowledge primarily through two sources--using past context to fill in missing information and/or asking the user directly for it [WALTZ78] [CODD78]. I want to introduce the use of world knowledge as a third way of obtaining such information. At the same time I want to use the world knowledge to detect requests not answerable with data available in the data base.

IV.3 Purpose of Problem Solving Frames

Collecting together such information still does little for bringing the system closer to developing a plan for answering the request. The problem solving frames are to propose general techniques such as problem reduction (reducing a problem to simpler subproblems)--for solving problems. To be able to develop a set of problem solving frames that classify problems and techniques for solving them, we must categorize the "kinds" of problems encountered in our data base environment (based on the assumption that the data base world restricts us enough so that the number of interesting classes of problems is manageable). Our studies have shown that the following kinds of requests occur most often: statistical analyses (correlation of data, etc.), categorization (classifying data into categories), association of data types with each other, ranking ("top five", etc.), plotting, causality, very general inferencing and operations on sets.

Another major contribution would be the ability to provide different "views" to a problem and to the values stored in the data base. In essence an "overlay" of the data base could be generated for a particular problem. An example would be data stored in a daily format in the data base viewed as if weekly data existed. This same mechanism could be applied to bring two seemingly disjoint concepts in a problem together.

IV.4 A Scenario

A sample scenario of the use of problem solving frames can be seen in Figure 2. They are activated by key words and phrases in the user's request and as side-effects to the instantiation of particular frames during the semantic analysis stage of JETS. The activated frames analyze the request to decide if they can provide any guidance in formulating a plan capable of rendering a solution to the problem. If no such assistance can be offered, the frame is deactivated; otherwise control is passed to that problem solving frame. Figure 3 gives an example of a problem solving frame for comparison.(*) Notice the use of production like rules that associate specific conditions and actions. These actions include drawing in other problem solving frames and invoking specific functions such as MAKE-UNITS-EQUAL which tries to convert the units of measurement (time, length, etc.) of the individual fields into equivalent forms. Figure 4 demonstrates this concept by exhibiting the hierarchy of unit frames and by presenting an example instantiation for the phrase "... 54 minutes ...".

In summary, the value of this type of frame is that it provides a general forum for taking all the information provided by the frames instantiated during parsing and semantic analysis and analyzing it so that a program can be generated to answer the question. In other words, it is not responsible for writing a program at the formal query level but only to decide what

* The frames in our system are actually being written in FRL but the examples in this paper were written in a more simplified form for readability.

```
U: Compare NORS and NORMUS percentages by squadron and by wing for May 73.

J: Invoking COMPARE frame.  Invoking PERCENT frame.

   NOR (Not Operationally Ready) not in  the data base.  It can be obtained by
   ADDING  NORS  (NOR  due  to  Scheduled  Maintenance)  and  NORMUS  (NOR due to
   Unscheduled Maintenance).

   SQUADRON  partitions the existing planes up.  No such data is stored in the
   data base.  Such partitioning can be done by:

      1 plane serial number
      2 plane type
      3 none of the above

Please select one of the above:

U: 2

J: WING linked to SQUADRON. PLANE TYPE absorbing WING.

   I have interpreted your request as follows:

     1 Retrieve NORS, NORMUS by plane types for time period of May 1973.

     2 Form NOR=NORS+NORMUS.

     3 Generate NORS%=NORS/NOR and NORMUS%=NORMUS/NOR.

     4 Construct table of PLANE TYPE, NORS percentage, NORMUS percentage.

     5 List by PLANE TYPE the maximum of NORS percentage or NORMUS percentage.

   Does this satisfy your request?

U: yes

J: Executing...

      PLANE TYPE   NORS%   NORMUS%
          A7         71%      29%
          F4         55%      45%
      SKYHAWKS       27%      73%

          A7: more NORS
          F4: more NORS
          SKYHAWKS: more NORMUS
```

A Scenario

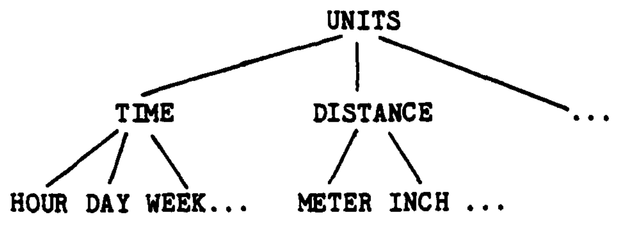
figure 2

```
Invoking Concept:
  COMPARE, MORE, LESS, CORRELATE, ANALYZE, ...

Context(s):
  Requires instantiation of two or more entities
  to compare:
    (1) one FIELD over two or more ranges, or
    (2) two or more different FIELDs:  FIELD1,
                 FIELD2,...

Actions:
 Context=(1):
  if FIELD.TYPE=num & CONCEPT=more
     then invoke greater;
  if FIELD.TYPE=num & CONCEPT=less
     then invoke less;
                .
                .
                .
 Context=(2):
  if FIELD1.TYPE=num & FIELD2.TYPE=num
                     & CONCEPT=more
     then invoke greater;
  if FIELD1.TYPE=num & FIELD2.TYPE=num
                     & CONCEPT=less
     then invoke less;
  if FIELD1.TYPE=num & FIELD2.TYPE=set
                     & CONCEPT=more
     then invoke cardinality-of-set
          invoke more;
  if FIELD1.TYPE=num & FIELD2.TYPE=num
                     & CONCEPT=compare
     then invoke find-relationship;
  if CONCEPT=associate
     then invoke check-correlation
          invoke find-relationship;
                .
                .
                .

Global Actions:  make-units-equal
```

COMPARE Frame

figure 3

Make-Units-Equal



E.g. "... 54 minutes ..."

```
MEASURE3
        ...
        event = time7
        ...

TIME7
        ...
        begin = +0
        end = +54
        unit = minute7
        action = make-units-normalized
        ...

MINUTE7
        ...
        normalized-begin = +0
        normalized-end   = +3240
        unit = second
        ...
```

Defining Units

figure 4

question to answer, to break the question up into a sequence of simpler requests if the question is too complex, to provide higher level mathematical/statistical analysis of the data returned, and to call on a formal query generator to generate the actual formal query.

IV.5 Handling Vague and Complex Requests

Vague and complex requests can be handled in a couple different ways. One way is by finding a problem solving frame that matches the request almost exactly. These are the type of problem solving frames not yet described. Each frame will consist, in essence, of a sequence of simpler commands, where each simpler command is either the sort of question which can be understood directly, or another command which can ultimately be broken up into a sequence of questions that can be answered. They will also be used to generate dialogue with the user when vague terms must be further explained before a formal query could be generated (e.g. defining "worst" for the particular user and question).

In JETS, this kind of problem solving frame is useful for report generation. The following describes what a problem solving frame for requests like "Does trend analysis of failure and maintenance rates for all aircraft differ significantly from the corresponding rates of new aircraft?" would have to do.(*)

1 define "differ significantly" (user-defined or system default),

2 retrieve for each maintenance action X whether it was scheduled or unscheduled and time since last maintenance on same system,

3 form maintenance rate for each maintenance action,

4 project trend of maintenance rates by linear regression,

5 calculate average mean time between failure,

6 apply (1) through (5) to new aircraft,

7 generate table of trend of failure rate and maintenance rate for all data vs. trend of failure rate and maintenance rate for new aircraft, and

8 compute standard deviations for the trend of failure rates and maintenance rates for all aircraft and for just new aircraft.

The problem solving frame writes all of the program required except the generation of the basic formal queries (the ones that retrieve the actual fields used to calculate the failure and maintenance rates) which are generated by the formal query generator [GOODMA77].

* Taken from list of questions most asked by 3-M Naval personnel[NALDA].

If no direct match is found in the problem solving frames, the system will use the second method to handle the vague and complex request. This method involves trying to turn the request into a well-defined problem. For example, we will consider the same vague and complex request as discussed above:

"Does trend analysis of failure and maintenance rates for all aircraft differ significantly from the corresponding rates of new aircraft?"

IV.5.a <u>How can the system attempt this problem?</u>

The ability of the system to handle requests of this nature revolves around its competence in locating and using pieces of problem solving knowledge coded into the frames and heuristics. Below is a sequence of steps the system might follow in <u>planning</u> how to solve the above problem and explanations of where they come from; I hope the steps give a better idea of the kinds of knowledge the system must contain. The order in which the items are considered is based upon the arrangement of the constituents in the sentence and due to heuristics and information in the frames.

(a) How should "maintenance rate" and "failure rate" be internally represented?
The system must determine what "maintenance rate" and "failure rate" mean in this context since the terms do not appear directly in the data base. "Rate" could possibly have at least four different interpretations: (1) mean time between events; (2) fraction of time a particular state is in effect; (3) percentage of times when <u>event1</u> occurs given the occurrence of some other <u>event2</u>; or (4) number of events per time period. The most likely interpretation for "failure rate" would be (1) followed, in order, by (4), (2) and (3). The right interpretation can be judged by noting that "failure" is an <u>event</u>, so (2) is unlikely since it involves <u>states</u>; no other event is listed nor can one be inferred easily, so (3) is unlikely (contrast "safe landing rate" which suggests, via safe, some other event (e.g. "unsafe landings" or "total landings")); (4) could be eliminated by a priority scheme favoring (1) or, since (4) is actually so closely related to (1), it perhaps doesn't even deserve a special status. For "maintenance rate", the best interpretation would probably be (1) (though (4) would be feasible, too). The decision of which interpretation to use could be deferred until other global factors (such as "what is trend analysis?") are considered. One nice feature of interpreting "rate" in this way is that it makes it easier to decipher the meaning of other rates such as <u>landing</u> rate (definition (4) seems to fit best), <u>crash</u> rate (definition (4) fits best), <u>safe landing</u> rate (definition (3) seems best as mentioned above), etc. Another important consideration here is that a "failure almost always" results in a "maintenance" so the system might want to omit any maintenances that were due to failures (i.e. unscheduled maintenances) from the list of maintenances when computing "maintenance rate".

(b) How will "mean time between failure" and "mean time between maintenance" be computed? (assuming these were chosen to represent failure and maintenance rates, respectively)

This will probably be the most difficult step in the plan for the system to come up with (in fact this one part of the overall plan is a complex problem in its own right). This is the point where what is available in the data base must be tied into the user's description of what is wanted. This involves breaking up the definition of "mean time between failure" and "mean time between maintenance" into components so that they can be calculated. Since no "mean times" are stored directly (and this fact would have to be discovered while examining the data base model), the system must calculate them by looking at each maintenance action on a plane, determining if the maintenance was scheduled or unscheduled (which implies a failure occurred), and recording the date of the maintenance. Standard deviations, etc. should also be calculated since the system may need them later. It would be desirable to keep some detailed data about this processing around for follow-up questions. The system should also consider saving the quantities calculated in partitioned sets, by time, plane serial number, etc. It might turn out that the <u>overall</u> failure and maintenance rates may be all that will be needed, but the system can't know for sure at this point. Both options should be kept open, in the hope that other information will aid in choosing one or the other interpretation. Otherwise the highest priority or lowest cost interpretation could be chosen.

(c) What does "all aircraft" designate?

The system must decide if it really wants to look at <u>all</u> aircraft for <u>all</u> dates or to just consider a sample. This might be a place where the user should be requested to be more precise (especially should cost-estimates show that the search over all aircraft for all dates is too expensive). Other possibilities include basing the statistics gathered on a small group of "typical" aircraft selected for use in problems like this one. Later the system may have to come back and consider whether "new planes" should be included in "all planes" or does "all planes" mean "all except new planes".

(d) What is "differ significantly"?

The best case of all would be if the decisions made at the time when "mean time between failures" and "mean time between maintenances" are handled set up expectations for appropriate tests of significance. Those expectations would make it easier to choose from among the following possible definitions of "differ significantly": (1) given two numbers, check for the percentage by which they differ (not precise, but fits our common-sense interpretation); (2) given two groups of numbers, compute their means and check to see if they fall within two and one-half standard deviations of each other (or some standardized statistical criteria); or (3) given two sequences of data, check to see if they are correlated.

(e) What does "corresponding rates" refer to?

This will most likely be interpreted by the parser and semantic interpreter. The difficulty is that the system will have to realize that "corresponding rates" refers to the earlier rates--failure and maintenance. The examination of a general knowledge frame describing "correspond" would help resolve this.

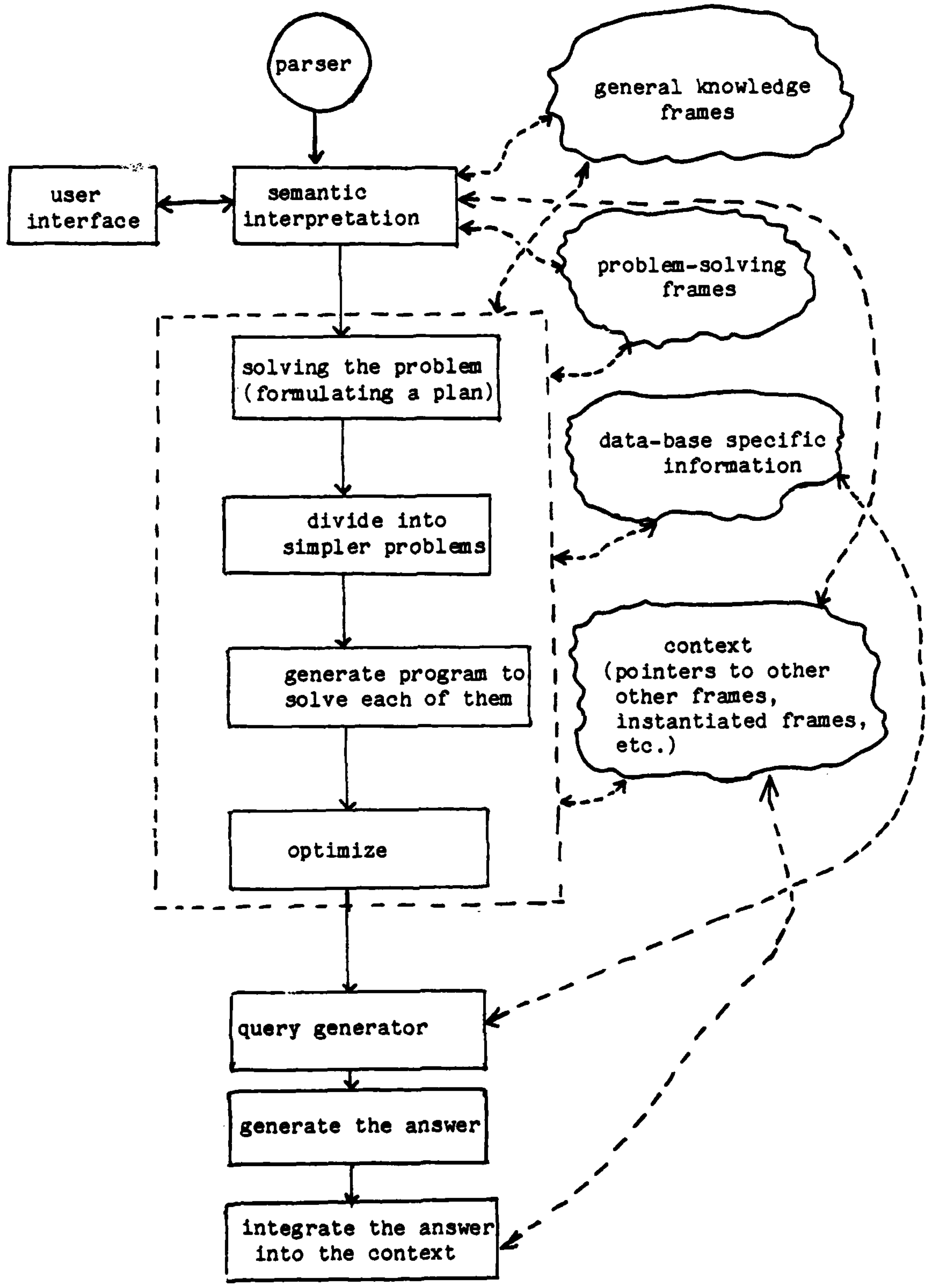
(f) What are "new aircraft"?
The system must decide how far in time should go from the date an aircraft begins service and still consider the aircraft "new". The system could pull in a general knowledge frame that defined "new" or could ask the user directly "how new is new?".

(g) What is "trend analysis"?
The system can examine its general description of trend analysis and decide what particular method (e.g. linear regression) should be applied here. Trend analysis involves the projection (or interpolation) of a sequence of numbers over time. Before it can be performed, it is necessary that a sequence of numbers be collected. This ties in with the "failure and maintenance rates" except that trend analysis needs a sequence instead of just two numbers. This might suggest the further partitioning of "failure and maintenance rates" by how old an aircraft is (e.g. group together all "one year old" aircraft). Hence an attempt to fill a descriptor in the "trend analysis" frame has propagated back to cause us to prune and reorder our options (<u>instead of backtracking</u>) for looking at an object that we had previously interpreted. This demonstrates the use of problem solving frames to help turn a problem into a <u>well-defined</u> problem.

(h) How should the answer be displayed?
Heuristics could help decide when it would be useful to generate a table (graph, etc.) to display the data returned. The question actually asks a yes/no answer, but clearly this alone is inappropriate.

(i) What information should be saved for follow-up questions?
The lower level information that will have to be retrieved from the data base might be useful in answering follow-up questions about specific areas of the original problem. Must decide what parts of the data should be saved and how it should be grouped. The hope is to avoid duplicating searches of the data base for information that was just retrieved in an earlier query.

A diagram showing the structure of JETS can be seen in Figure 5.


parser
general knowledge frames
user interface
semantic interpretation
problem-solving frames
solving the problem (formulating a plan)
data-base specific information
divide into simpler problems
context (pointers to other other frames, instantiated frames, etc.)
generate program to solve each of them
optimize
query generator
generate the answer
integrate the answer into the context

An Overview of JETS

figure 5

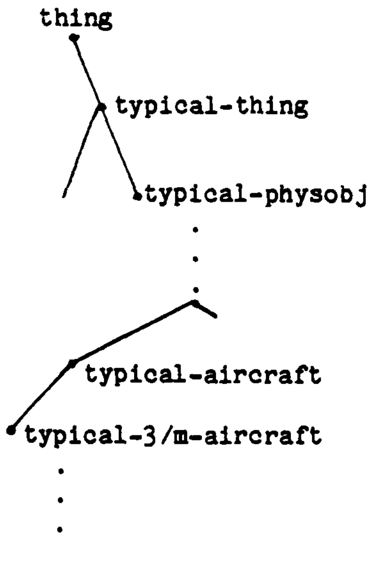
V

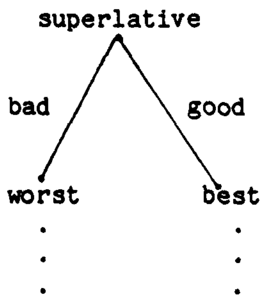
The Approach

V.1 Hierarchy of Frames

The problem solving frames are general descriptions of problems (and solutions). Much more power could be milked from the concept of problem solving frames if the frames were loosely tied together in a hierarchy network. This hierarchy of problem frames would work as follows. The system of frames start at a very general level describing the nature of the problem environment and proceed on down to the particular domain in question. The information found by threading one's way down the hierarchy can be used to help develop a mapping between the general problem solving frames and the user's description of his or her problem.

Much of the work involved in solving a problem is spent trying to define "imprecise" terms in the problem statement itself. Such terms include "best", "worst", "typical" and so on. While in many cases the system will have to consult the user directly to define such terms (*), a separate hierarchial structure describing the meaning of these terms can be constructed and linked into the problem solving frames hierarchy. Below are a few segments of such a hierarchy showing the names of concepts that might be represented in it.





(*) These definitions can be saved under a particular user's name for future reference--allowing us to develop a model of the user.

V.2 Solving the Problems

The basic description of the kinds of representation structures for problems was found in the last chapter (e.g. in the COMPARE frame example). The value of such structures, however, relies heavily on the ability to relate descriptions of specific problems to the general descriptions referred to in the problem solving frames. The need for very general and powerful matching routines can be seen if the problem solving frames are going to work. The matcher must find matches between an element of the user's depiction of a problem and an element of the models of problems by trying to "fill" the slots of the model. Items from the user's problem descriptor, which satisfy slot descriptors of the problem models, fill the slots and can be said to match the slots. The matching routines must find when "gaps" exist between a general description of a problem environment and an actual user description of a problem--extracting information from the knowledge base, past context or through dialogue with the user to fill the gaps.

Several problem solving techniques can be employed during the matching process. These include analogy, decomposing and recombining, and heuristic reasoning. Analogy involves trying to find a mapping between the set of problem descriptions and the user's actual request. If an almost perfect match exists, analogy is reduced to a task of indentification--identifying the problem with known solutions. The analogy process can be divided into three phases: map, solve and lift [BROWN].

(1) Map

When the user's request cannot be solved immediately (because the problem is too complex or incomplete), it is necessary to try to map the problem into an analogous problem described by a set of problem solving frames. The process first involves summarizing the user's problem, leaving out the specifics, so that it is easier to explore the problem solving frames for possible partial matches. When matches are found, a simple mapping can be constructed. This mapping must be extended to include some of the details left out of the summary of the user's problem. When the map is finally finished, it can be employed to fill in the slots of the problem solving frames with the details of the original problem (the values going into the slots might undergo transformations during the mapping process as well). The result of this will be an analogous image problem.

(2) Solve

The set of problem solving frames activated during the map process can be used to solve the analogous problem during the solve process. Since the problem solving frames not only describe problems but also ways of solving them, the solution techniques described by them should be put together to solve the problem. If no solution is found (and this is possible when the problem is mapped onto several problem solving frames whose solutions might possibly compete against each other), then the analogy map must be changed or the analogous problem must be altered by adding further details. When a solution is found, an inverse map is applied to it. If feasible, the answer technique that makes up the solution is checked to see that it works.

(3) Lift

If the solution technique works, then we need to "lift" the solution from the problem solving frames back into the user's domain. This process is done by taking the plan constructed for solving the problem and, for each single step of the plan, applying the inverse map to it. It is possible that some parts of the plan are incomplete after being mapped back into the user environment. If this occurs, an attempt is made to patch up that part of the plan. If this fails, another analogy must be tried. If all parts of the plan can be lifted, then the plan is lifted, applying patches as necessary so that the plan fits in the user domain.

Decomposition is the technique of dividing the problem into subproblems that are (hopefully) easier to solve[POLYA57]. In reality, the hope is that problems can be reduced to simpler subproblems repeatedly until primitive (but solvable) subproblems are reached. Once this is achieved, it is easier to get a solution for the original problem. This fits very naturally into the hierarchy of problem solving frames environment because the frame system itself divides up components of a problem into simpler conceptual structures. These structures often correspond directly to the more primitive problems referred to in the decomposition process. The opposite process, which must be used after the success of decomposition, is recombination--putting together simple problems (or solutions) so that the principal problem can be solved. Recombination could also be a principal force in solving a problem. A user might input his problem in several steps. In that case, it might be necessary to combine the individual steps into one problem when looking for a match in the problem solving frames (i.e. map from many to one instead of the normal one-to-many).

Heuristic reasoning involves making intelligent stabs at solving a problem when the other techniques cannot be used. For the most part, heuristics will be used to try to get the problem description into a form so that one of the other problem solving techniques applies. They can, on their own, sometimes suggest how to solve a problem. The PLANES query generator [GOODMA77], which was based entirely on a set of domain-specific and domain-independent heuristics, was able to handle the majority of simple requests.

An example showing the use of these techniques can be found in Figure 6 below.

V.3 Handling Incomplete Problems

The detection of incompletely-specified problems is an important task in a problem solving world. Once it is detected that some information is missing, the problem statement can be brought up to par by:

(1) searching recent context;

(2) using world knowledge; and/or

(3) asking the user directly to fill in the gaps.

```
Problem:  Constrast NORS and NORMUS percentages by squadron
           and by wing for May 73.
              |
              |transformations done in semantic analysis
              |and beginning problem solving (actually,
              |ouput would be set of filled-in frames
              |and not an English sentence like below)
              V
          Compare [NORS/(NORS+NORMUS)] and
           [NORMUS/(NORS+NORMUS)] by plane type for
           May 73.

Frames instantiated:

   user problem descriptor frame
          ...
          operation  =  compare
          object1    =  percent1
          object2    =  percent2
          restrict1  =  planetype1
          restrict2  =  time1
          ...

   percent1                          percent2
    ...                               ...
    oper1 = NORS1                     oper1  = NORMUS1
    oper2 = NOR                       oper2  = NOR
    operation = oper1/oper2           operation = oper1/oper2
    type  = numerical                 type   = numerical
    ...                               ...

   nors1                             normus1
    ...                               ...
    English   = Not Operationally     English   = Not Operationally
                Ready due to                      due to
                Scheduled Maintenance             Unscheduled
                                                  Maintenance
    concept   = maintenance           concept   = maintenance
    type      = numerical             type      = numerical
    subsetof  = NOR                   subsetof  = NOR
    ...                               ...

   nor
    ...
    English   = Not Operationally Ready
    concept   = maintenance
    type      = numerical
    madefrom  = NORS+NORMUS
    ...
```

Figure 6, p1/5

```
   planetype1
    ...
    class    = set
    type     = (A7,F4)  ;only aircraft in our data base
    serialno = unknown
    ...

   time1
    ...
    begin    = (00:00, May 1, 73)
    end      = (23:59, May 31, 73)
    unit     = date

COMPARE Problem Solving Frame:

 ...
 AKO           = {OPERATION,ACTION}
                     ;COMPARE is a-kind-of operation
                     ; and action
 COMPARE-ELEMENTS =
      ($REQUIRE
         (#ELTS > 1)
            ;i.e. more than one element needed
            ;for compare
         (∀ i<>j, elt(i)<>elt(j) )
            ;each element must be different--if
            ;not different must make different by
            ;adding restrictions such as over what
            ;time period the element is to be
            ;considered
         .
         .
         .
          )
COMPARE-TYPE =
      ($REQUIRE
        (  TYPE ∈ {CORRELATE,ANALYZE,MORE,LESS
           COMPARE,...}))
      ($DEFAULT COMPARE)
            ;if no type specified, default
            ;to TYPE=COMPARE
      .
      .
      .
       )

  ...
```

Figure 6, p2/5

```
   INVOKE COMPARE
      Scanning through the problem solving frames, a
      match was found between OPERATION=COMPARE and
      FRAME-NAME=COMPARE.  A mapping between the
      COMPARE frame and the user problem descriptor
      frame will be attempted.

  USER ENVIRONMENT      --------->    COMPARE FRAME

OPERATION=COMPARE                  COMPARE-TYPE=COMPARE
                                   Checking COMPARE {...}.
                                   O.K.
OBJECT1=PERCENT1                  (CONS 'PERCENT1 COMPARE-ELEMENT)
                                   Checking...cannot perform
                                    checks.  Stacking.
                                    {PUSH CHECKLST
                                     "compare-elements: (#elts>1)"
                                     "compare-elements:
                                       (∀ i<>j, elt(i)<>elt(j) )"}
OBJECT2=PERCENT2                  (CONS 'PERCENT2 COMPARE-ELEMENT)
                                   Checking...
                                    (#elts>1)...O.K.
                                    (∀ i<>j, elt(i)<>elt(j))...O.K.
        .
        .
        .

   Provisional map completed.          [NOTE: might implement the
   Popping CHECKLST.                     CHECKLST as an agenda
    Checking...                          where constantly try to
     Compare-elements: (#elts>1)...O.K.     test the restriction]
     Compare-elements: (∀i<>j, elt(i)<>elt(j) )...O.K.
        .
        .
        .

  Phase completed.

Now that a map has been completed, will try to solve the
"mapped problem" using information provided in other companion
frames to the COMPARE frame.  These frames will not be shown
here but they include such frames as MORE, LESS, CORRELATE,
FIELDNAME, etc.
```

Figure 6, p3/5

They suggest the following solution sequence:

```
 1. Retrieve percent1.
 2. Retrieve percent2.
 3. Percent1>Percent2?   (because percent1 and percent2 have
                          to have numerical values, ">" was
                          chosen).
```

Will now try to find an inverse map to map this back onto the user problem.

The inverse of the original map can be applied to each of the above steps. Patches may be necessary for it to work.

```
Retrieve percent1.----------> Retrieve (object1=)percent1.
                      Patching... Apply restrict1.
                                  Apply restrict2.
                              Retrieve (object1=)percent1 for
                               (restrict1=)planetype1 and
                               (restrict2=)time1.
                      Patching... Expand percent1.
                      Patching... Decomposing.
                              Retrieve NORS and NORMUS for
                               plane types A7 and F4 for
                               May 1973.
                              Form NOR = NORS+NORMUS.
                              Form percent1 =NORS/NOR.
Retrieve percent2.----------> Retrieve (object2=)percent2.
                      Patching... Apply restrict1.
                                  Apply restrict2.
                              Retrieve (object1=)percent2 for
                               (restrict1=)planetype1 and
                               (restrict2=)time1.
                      Patching... Expand percent2.
                      Patching... Decomposing.
                              Retrieve NORS and NORMUS for
                               plane types A7 and F4 for
                               May 1973.
                              Form NOR = NORS+NORMUS.
                              Form percent2 =NORMUS/NOR.
Percent1>Percent2?----------> Is percent1 more than percent2?

                             O.K.
```

Figure 6, p4/5

Patching...Recombining..Smoothing...
[smoothing will be done by numerous heuristics; how well it works depends on the quality of the heuristics]

1. Retrieve NORS and NORMUS for plane type A7 and F4 for May 1973.
2. Form NOR = NORS+NORMUS.
3. Form percent1 = NORS/NOR.
4. Form percent2 = NORMUS/NOR.
5. Display table of NORS, NORMUS, NOR, PERCENT1, PERCENT2, PLANETYPE, and DATE.
6. For each row, is percent1>percent2?

A Simple Example of Problem Solving

figure 6

The actual detection that a problem is INCOMPLETE is harder than the gap filling process just described. Almost any problem in our domain at least "suggests" a solution by stating what is wanted. If one can "zero in" on what is requested, then drawing in all world knowledge relating to it should help detect if relevant information is missing. This might be a place where a mini-production system can be used. By putting the instantiated frames in a working memory (i.e. short-term memory), a general set of rules could then fire and try to fill in the gaps. Also drawn in by the rules would be so-called "expert problem frames" (in reality they would be the more general frames in the hierarchy of problem solving frames) that indicate the minimal kinds of things required to solve this particular class of problems.

For example, consider the input "Compare flights." and assume that no past context exists. In light of the last example, you can see that this input is incomplete because the COMPARISON task requires a minimum of two entities for a comparison to be performed. The problem solving frame COMPARISON (and other frames beneath it in the hierarchy) and the semantic frame FLIGHT would be drawn in. A set of production rules would fire that read that the minimal requirements for the COMPARISON frame to be activated is that two or more entities be available for comparison. Since "flights" does not satisfy this requirement, the FLIGHT frame would be scanned to see if there is a way to expand it to two or more entities. It would find that "flights" could be partitioned by any combination of plane series types, plane serial number, time period, and so forth. If past context was available, a search backwards for such partitions could be initiated. In this case, since no past context exists, the user would be asked to partition "flights" using one or more of the mentioned entities.

V.4 Assimilation

In the particular case of JETS, it would be beneficial if the "answers" themselves to particular queries by the user became part of the knowledge base, too. This way a user could refer directly to the last answer--either by stating some of the information returned or by stating "...from the last answer..."--in formulating the newest request. I plan to integrate the answer into the knowledge base in two forms. The first would be generating a set of frames that describes the answer in total. This would be done by linking the actual values from the answer into the frames instantiated during the statement of the problem. Should the answer be too long or involved, a second method would be used: instead of putting the actual answers into the frames, a description of the answer and pointers to a set of temporary files that contain the actual data returned would be employed.

V.5 Efficiency

Models of both the relational data base and the formal query language could be used to allow more efficient solutions to be generated. The structure of a data base (in terms of how the relations are formed) and the capabilities of the query language (in terms of being able to avoid duplication of searches in multiple relation requests, built-in functions, etc.) can strongly influence how hard it is to retrieve certain data types. The data base model could contain a hierarchial description of the relationships between different fields and relations and definition of non-exisitent fields that can be created from existing fields. It should provide data base "views" for different users ("overlays") and provide advice on efficiency considerations in search strategy selection (e.g. which relations provide best filter so that in a multiple relation query can search the relation providing the best filter first; heuristics that take into account collected statistics on the number of data cards in each relation, etc.). The query language model could be designed to allow easier movement to a new data base, easy change to a new query language and easy addition of new features to the current query language. In summary, the data base and query language models' principal goal is to keep the natural language portion of JETS from having to know anything about the query language or the structure of the data base.

VI

Conclusion

In conclusion, I would like to summarize the principal goals of my work:

1 to classify the kinds of problems occurring most often in a natural language environment (and in particular, in our user environment),

2 to find structures to represent general descriptions of problems and solutions,

3 to develop powerful routines to match these general problem descriptors to user requests,

4 to generate ways to solve the problem, and

5 to integrate the answers back into the general knowledge base for future reference.

During this process, I hope to be able to have the system generate English descriptions of its plans for solving problems.

APPENDIX A

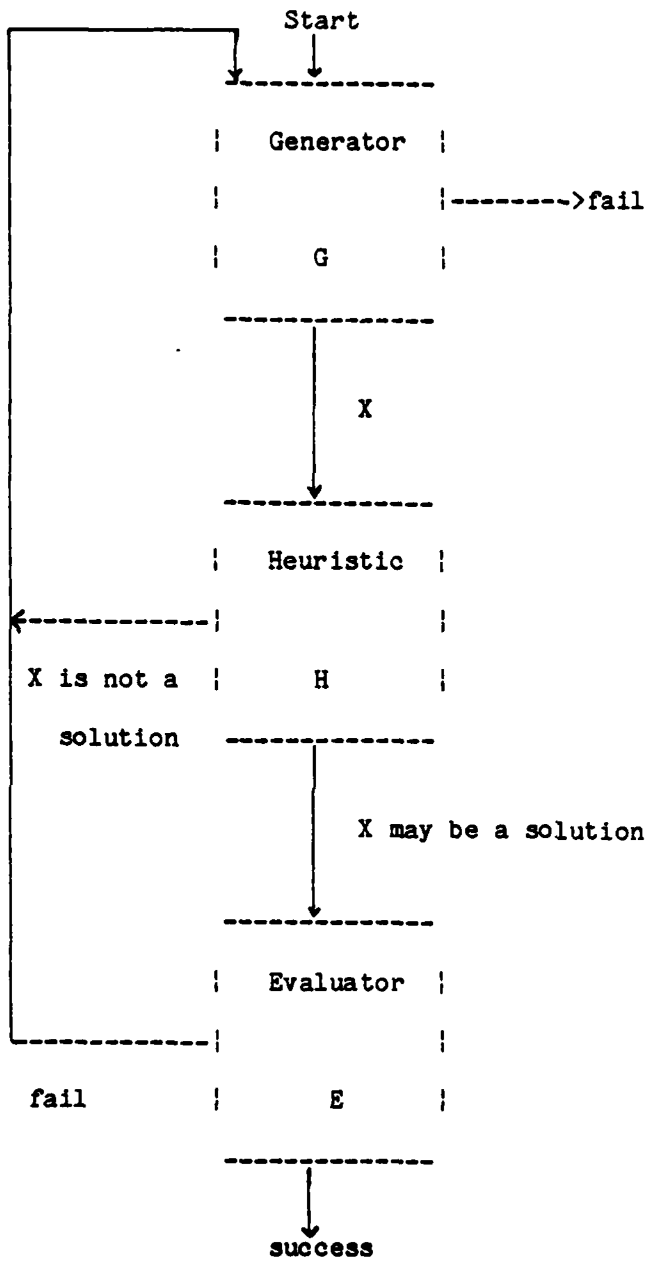
Historical Look at Previous Representations for Problems

A.1 State Spaces and Problem Reduction Operators

A problem solving task could be defined as a graphical structure consisting of a set of nodes--representing the possible states--and a set of links--representing the possible operations that can be applied to the states to transform them into a different form. The hope is that the application of an operatior to a description of the current state will reduce the problem to a new state that is hopefully closer to the goal or to subproblems that can also be solved. Such a process can be performed by relatively automatic procedures. What is generated during this process is called a "search tree". A major difficulty with such a searching process is the combinatorial explosion of possible paths in a search tree that can result. This occurs because the application of the operators to the current state can result in as many new states as the number of operators. Thus, continuing in this fashion for very long can result in an extremely "broad" search tree. To help cut down on the number of nodes investigated in the tree, heuristics can be employed to suggest which nodes to investigate first (because those nodes seem most likely to be closer to the solution) and to prune nodes that cannot lead to a solution. The use of heuristics can lead to another problem called the "Horizon effect" [BERLINER]. The horizon effect occurs when the heuristics select certain nodes to investigate over others because they see short-term gains. Those short-term gains, may not, in the long-term, have gotten you any closer to the goal. In fact, investigating them may have kept you from following a path to the goal.

A.2 Generate and Test Paradigm

A problem could be thought of as: (1) a TEST for a class of symbol structures (the solutions of the problem), and (2) a GENERATOR of symbol structures (the potential solutions). To solve a problem would mean to generate a structure, using (2), that satisfies the test of (1). If the problem cannot be solved directly, it is often possible to search for a solution. For intellectually difficult problems, the number of possibilities to be searched is so large (sometimes even infinite) that for all practical purposes there is no exhaustive procedure. Since this closely parallels state spaces, heuristics can be employed to define and modify the search. Thus a heuristic serves as a FILTER interposed between the solution candidate generator and the solution candidate evaluator. This is shown in figure 7.



Generate and Test Paradigm

figure 7

A.3 Predicate Calculus

First-order predicate calculus is a formal language that can express much of mathematics and many English statements. It has associated with it a small number of inference/deduction methods for making valid logical deductions of new statements from a set of given ones. Thus the concept of separation of "data" and "programs" is clearly embodied. A problem solving task involves trying to "prove" that the statement is true. This is normally done--instead of proving it directly--by showing that the negation of the statement is false. The tree, similar to search trees, is constructed during the process and a search is made until an invalid statement is generated.

Programs have been written that try to translate English into predicate calculus formalism. There are two major difficulties that occur here: (1) deciding which connectives to use when translating the English sentence, and (2) the unambiguous transformation of the intended meaning of the sentence into a predicate calculus statement. Ambiguities in English often can only be resolved by referring to the context in which the sentence appears. Adding this context to the predicate calculus system would be extremely difficult and awkward.

A few attempts have been made to try to express subsets of English using predicate calculus formalisms. I will discuss one of the systems that was developed. QA3 was a natural language understanding system that could accept a restricted subset of English used to describe a wide variety of topics [GREEN]. The system used first-order logic as its language and a theorem-prover as its deductive mechanism. The goals followed while creating QA3 include: (1) finding a language general enough to represent reasonable questions, answers and data about the domain; (2) the ability to search efficiently the stored data for relevant information for a particular request; (3) the ability to derive an answer to a question even when the answer is not explicitly stored; and (4) the ability to easily add or delete facts from the data. So QA3's knowledge of the world is expressed as a set of axioms and the questions asked of the system are presented as theorems to prove. For example, the fact "CSL is a building" could be encoded as the axiom "BUILDING(CSL)". The question "Is CSL a building?" could be formulated as the theorem to prove "BUILDING(CSL)". The kinds of facts and theorems encoded can be of varying complexity, as can be seen in the example below.

```
Fact: A robot is a machine.
STATEMENT: (∀x)[ROBOT(x)==>MACHINE(x)]
Fact: Rob is a robot.
STATEMENT: ROBOT(Rob)
Fact: No machine is an animal.
STATEMENT:  (∀x)[MACHINE(x)==> ANIMAL(x)
Question: "Is every thing an animal?"
          (∀x)ANIMAL(x)
Answer: No, x=Rob.
        The answer indicates ¬ANIMAL(Rob) is a theorem.
```

The actual construction of answers utilizes the Resolution method of proving theorems. The resolution method is simply a method of proof by contradiction.

One assumes that the negation of the theorem to be proved is true, and tries to find a contradiction from the negation and the original premises.

QA3 exhibits the weaknesses found in predicate calculus based question-answering systems mentioned earlier: the trouble with deciding what connectives to use and the inability to handle ambiguous input. It also shows a problem that is central to such systems--namely deciding the meaning of the objects of the system. When we have a predicate such as "BUILDING(CSL)", we have asserted something about "CSL". However, we do not really know what it means for "CSL" to have the attribute "BUILDING". My feeling is that the predicate calculus formalisms stop too soon in defining the meaning of concepts. Any natural language system that can hope to ever handle the complexities of everyday English must go deeper in defining the meaning of objects, actions, etc.

APPENDIX B

Other Approaches to Problem Solving

B.1 Expert Problem Solvers

It would be useful at this point to mention the specialized problem solving systems designed to work in one environment. These systems are designed to contain a knowledge base of all the important facts in a domain. Existing systems have expertise in organic chemistry (DENDRAL [FEIGENBA]), medicine (MYCIN [SHORTLIF]) (INTERNIST [POPLE]), and financial consulting (TIERESIAS [DAVIS76]). The TIERESIAS system is a good representation for the state-of-the-art today. It was developed in the context of MYCIN as a tool for constructing knowledge bases for expert systems. One of its applications was a financial consulting system.

TIERESIAS constructs a large data base of production-rules that encode specific knowledge about the domain. A numerical scale is used to indicate the "worth" of the suggestion made by the production rule. A sample production rule created with TIERESIAS for the financial consulting program follows (the production rule is not actually encoded in a natural language form).

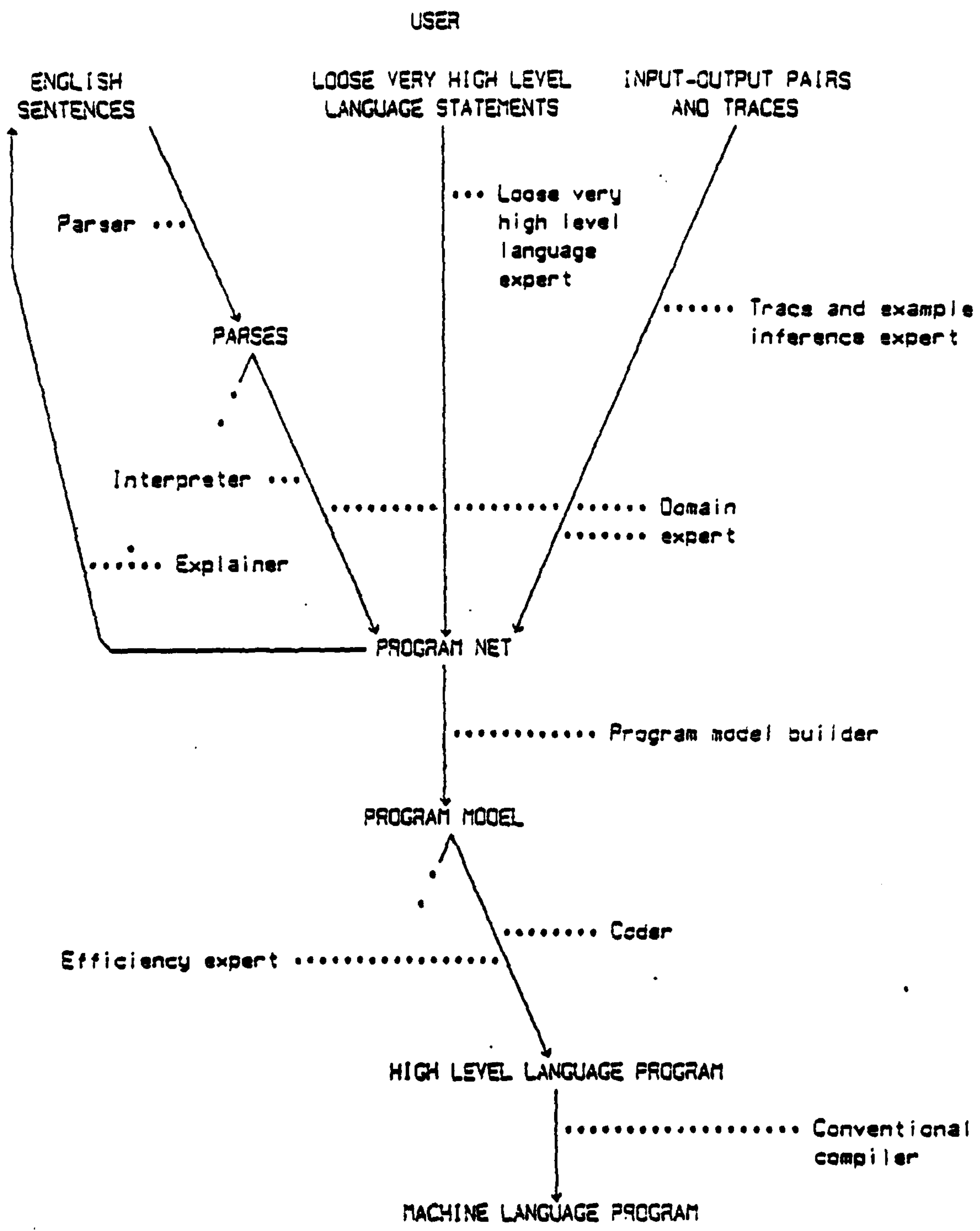
```
If  1- THE DESIRED RATE OF RETURN ON THE INVESTMENT
       IS GREATER THAN 10%
    2- THE AMOUNT OF INVESTMENT EXPERIENCE OF THE
       CLIENT IS MODERATE
    3- THE AREA OF THE INVESTMENT IS NATURAL
       RESOURCE DEVELOPMENT
then
    1- THERE IS EVIDENCE (.5) THAT THE NAME OF THE
       STOCK TO INVEST IN IS GEORGIA PACIFIC.
```

The system works by initiating a dialogue (i.e. a case study) requesting information for its short-term memory until production rules begin firing giving suggestions or requesting more specific information. For adding rules to the system, TIERESIAS guides an "expert" in a menu-driven dialogue to get useful information. When it encounters "unknowns" in the rules, it uses "schemas" to try to figure out what an unknown is or forces the expert to define the unknown from knowns. The schemas are simply definitions of concepts such as "stock name" (where Georgia Pacific is a specific stock name). One of the most interesting problem solving aspect of Davis' work is his use of "knowledge about the representations" and "knowledge about knowledge" itself (META-KNOWLEDGE). He actually encodes production rules that know general things about other production rules and cause those rules to be updated depending on changes to the knowledge base.

B.2 PSI: A Program Synthesis System

Another approach in problem solving are the automatic programmers. These are systems that accept an input describing how a simple program can be written to perform a required task and output a program to perform that task. The PSI system [GREEN] can write a program in LISP if it is given step-by-step English descriptions of the program steps. This description should employ little "internal knowledge" so that PSI can generate high level specifications of what the user wants done. In fact, PSI relies heavily on the use of knowledge about programs in general instead of on problem solving. This knowledge on programs is used to tie "low level" descriptions of the program together (i.e. looks at request by user as representing several lines of a program but not necessarily in the correct order).

PSI's operation can be broken into two phases: the acquisition phase, which acquires the model of the program; and the synthesis phase, which produces a program from the model. Sentences are first parsed, then interpreted and finally stored as a program specification. The parser is a general English parser that utilizes knowledge about English usage. The interpreter knows much about program synthesis, using both knowledge about programming and about the topic under discussion. A dialogue moderator is used to guide a dialogue with the user--selecting or suppressing questions. It tries to keep the user and PSI from straying away from each other. The program specification and program model are major parts of PSI. Both are high level program and data structure description languages. The program model contains completely specified high level algorithm and information structures. The program net, on the other hand, forms a loose program descriptor. An overview of the PSI system can be seen in Figure 8 below.



Overview of PSI

figure 8

APPENDIX C

Knowledge Representation Systems

C.1 FRL

FRL [ROBERTS] is a facility for constructing and using frames. It consists of a template for a specialized data structure (the frame) and a collection of LISP functions for defining frames, storing and retrieving information. The sub-structures of a FRL frame are slot, facet, datum, comment and message. A description of an FRL frame is shown below.

```
(frame1
  (slot1 (facet1 (datum1 (label1 message1 message2 ... )
                                   ... more Comments ...)
                 (datum2 ...                           )
                 ... )
         (facet2 (datum1 (label1 message1 message2 ...) ... ) ...)
         ... )
  (slot2 (facet1 (datum1 (label1 message1 ...) ...) ...) ...)
  ... )
```

Slots are composed of facets. Facets are typically such things as $VALUE, $DEFAULT, $IF-ADDED, $IF-REMOVED, $IF-NEEDED, $REQUIRE, etc. $VALUE simply indicates that the data that follows in the list represents particular values of the slot. $DEFAULT tells what values can be assumed if no particular values have been assigned to the slot under the $VALUE facet. $IF-ADDED is a demon that is triggered whenever a value is put into a slot. $IF-REMOVED is activated when a value is removed from a slot. These demons are useful because they can do the "housecleaning" necessary when values are added or removed from a slot. Their activities include such things as changing values in other slots to reflect the changes in the current slot. $IF-NEEDED is activated when a value is requested from a particular slot but none is there and no default value has been provided. It can ask the user directly for a value or can construct a default value from data provided in other slots. The $REQUIRE facet is used to make sure a value assigned to a particular slot satisfies particular requirements assigned by the user (such as the value must belong to a particular class, etc.). Comments and messages can be tacked to values assigned to slots to provide more specific information about the value that might be needed later on. A sample FRL frame is shown below.

```
(fassert CSL
    (ako ($value (BUILDING)))
    (english-term ($value (Coordinated Science Laboratory)))
    (purpose ($value (research)))
    (staff-member ($if-added (payroll))
                  ($if-removed (payroll))
                  ($require
                    ((memq (fget :value ´degree)
                           ´(BS BA MS MA PHD)))))
    (floors ($value (6)))
    (computers ($default (DEC10)))
)
```

The FRL frames can be built up into a large tree structure using two FRL system functions as slots: AKO (A Kind Of) and INSTANCE. These provide paths for data to be inherited. AKO and INSTANCE are actually inverses of each other (so defining a relation with one of them automatically defines another inverse relation with the other). The resulting hierarchy of frames that is derived from this establishes a distribution of information with the more general information stored higher up in the hierarchy where it can be inherited by more specialized concepts lower in the hierarchy.

C.2 KRL

KRL [BOBROW] [TENNAN80] was designed to try to model how humans represent and use information, thus giving it a more psychological foundation than FRL. As in FRL, knowledge in KRL is organized around conceptual entities that have associated descriptions and procedures. The descriptions can represent partial knowledge about the entity and allow multiple descriptors so that the entity can be examined under different points of view. A PROTOTYPE is an object that is used as a basis for comparison with other entities in memory. It describes a class of concepts instead of a particular instance, i.e. particular objects are described through comparison to a prototype.

E.g.

```
(a Building with
   name = "Coordinated Science Laboratory"
   (an Address) = "Springfield Avenue")
```

"Building" is a prototype used in the description of a particular building.

There are several kinds of units in KRL ("unit" is simply the KRL term for "frame"). The units come from the set of category types: Basic, Specialization, Abstract, Individual, Manifestation, Relation, and Proposition. Different sets of KRL procedures are invoked depending on what category type a unit is. Basic units are simple mutually exclusive partitions of the world into different objects. No individual can be in two distinct basic categories. This makes it easy to decide if a particular object fits a description. Specialization units provide a refinement of a basic category. An Abstract unit describes concepts that are not physical objects. It is used to hold a set of descriptions and procedures that can be inherited by another entity that is defined by using the abstract unit as a prototype. The Individual units describe unique entities in the world that is being modeled. So EVENT137 in the example in Figure 9 below defines a unique event. Manifestation units seem to be a "catch-all" unit that can be used to define a particular demonstration of an individual in one context (i.e. from one point-of-view), to describe an individual using time-dependent descriptions without the need of creating another manifestation and to describe an individual whose unique identity is not yet known (e.g. "the murderer" instead of "John Doe"). The Relation unit provides a means of representing a relationship (or predicate) as an abstract mapping while the Proposition unit represents each instantiation of the relationship [TENNAN80].

The Match framework in KRL is superior to that of FRL because it allows limits to be placed on the search time and depth during the comparison process of different units. When these limits are reached, KRL can explain why it gave up the search and what state it was in at the time. As in FRL, procedures can be attached to the units to facilitate the matching and deduction process.

C.3 KLONE

KLONE[BRACHM78] differs from FRL and KRL because it embodies semantic nets as its knowledge representation formalism instead of frames. These semantic nets are simply sets of nodes with directed arcs. The nodes represent concepts and the links represent relations between concepts. KLONE tackles a problem that has been prevalent in the use of semantic nets to represent knowledge--the inconsistent definition of what a node and link can be [see WOODS75, pp. 35-81]. No attempt had been made at cleaning up semantic nets until Brachman and Woods' work [WOODS75] [BRACHM77]. Brachman provided a more rigorous foundation for defining concepts (represented by nodes) and associations (represented by links). He accomplished this with the use of primitives. For example, a special primitive link was provided to distinguish links used to define properties at concept nodes and those used to instantiate properties at instance nodes.

Throughout his design, Brachman apparently kept in view an ultimate goal of designing a knowledge representation system (KLONE) that brought extensional and intensional views of classes into focus. The extensional description of a class is simply a enumeration of existing and potential members of that class. It is used to capture the concepts of class membership and subset relations. The intentional description is more involved. It defines what it means to belong to a particular class. This is done by determining and making use of relationships between concepts independent of the particular objects they apply to. In other words, it describes the attributes necessary for something to be a member of the class and how these

```
[Travel UNIT Abstract                ...Travel is the unit name.  Its category type is Abstract .
   <SELF  (an Event) >                  ...description of the Travel unit itself.
                                        Event, Plane, Auto and City are known units
   <mode (OR Plane Auto Bus)>           ...either Plane or Auto or Bus
                                           can fill the slot named mode.
   <destination (a City)>]

[Visit UNIT Specialization              ... a specific category of SocialInteraction
   <SELF (a SocialInteraction)>
   <visitor (a Person)>
   <visitees (SetOf (a Person))>]

[Event137 UNIT Individual               ...a specific event described from two viewpoints
   <SELF  {(a Visit with
     visitor =  Rusty                ...The actor is the known unit Rusty
     visitees = (Items Danny Terry))    ...Items indicates at least  Danny and Terry are
   (a  Travel with                      set elements in this set
     mode = Plane
     destination=  SanFrancisco)}>]     ...SanFrancisco is a known unit describing a City
```

. KRL Representation of Rusty's Trip to San Francisco

Sample Unit of KRL[BOBROW, p.g. 10]

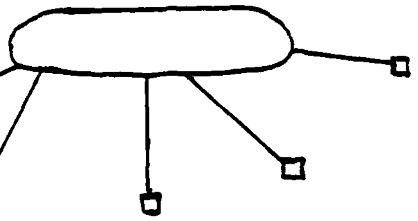
figure 9

attributes are related. E.g. "Coordinated Science Laboratory" is a building but what makes it a building is the fact that it satisfies a broad criteria such as providing shelter from the elements, being built out of physical materials, etc. [TENNAN80].

Out of Brachman's original work came the idea of "Structured Inheritance Nets" [BRACHM77] [BRACHM78]. This is where descriptions of structured conceptual objects are arranged in lattice-like networks and the inheritance between different descriptions is provided by "structured cables". The types of cables provided are:

(1) Satisifaction (INDIVIDUAL--->GENERIC)
(2) Restriction (GENERIC--->GENERIC)
(3) Differentiation (GENERIC-->GENERIC).

The "Individual" concepts describe individual objects and "Generic" concepts provide definitions of conceptual objects. Figure 10 shows what these structures look like and gives an example of a KLONE representation.

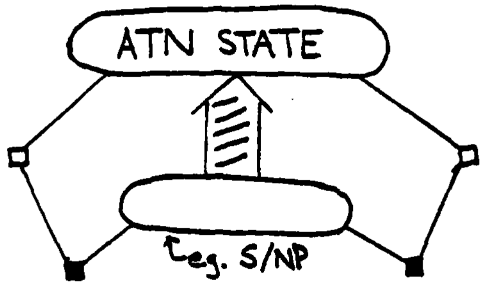


Generic Concepts

(generic) Roles

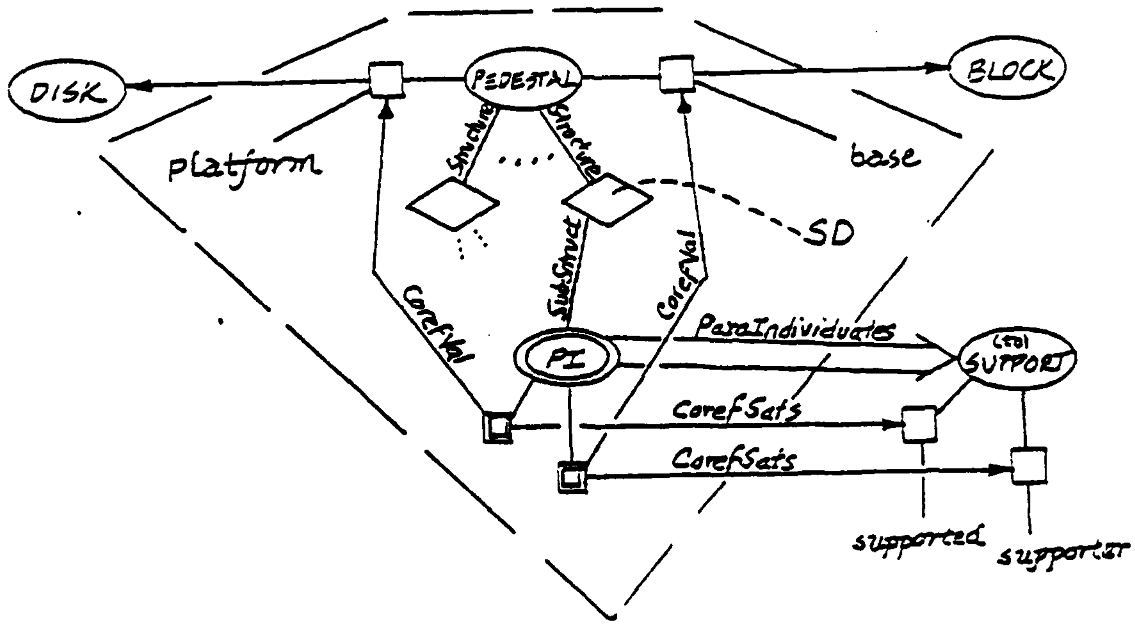
(Schematic templates--NOT prototypes or abstract individuals)

Individual Concepts



instance roles

(particularized descriptions of individuals)



KLONE Structures[BRACHM78, p.g. 33]

figure 10

VII

References

[BERLINER] Berliner, H. J.; Chess as Problem Solving: The Development of a Tactics Analyzer; Ph.D. Thesis, Department of Computer Science, Carnegie-Mellon University; March 1974;

[BOBROW] Bobrow, D.G. and Winograd, T.W.; An Overview of KRL, a Knowledge Representation Language; Cognitive Science, vol. 1, pp. 3-46; January 1977;

[BRACH77] Brachman, R. J.; What's in a Concept: Structural Foundations for Semantic Networks; International Journal of Man-Machine Studies, vol 9., pp. 127-152; 1977;

[BRACH78] Brachman, R. J.; Theoretical Studies in Natural Language Understanding; Report No. 3833, Bolt Beranek and Newman; September 1978;

[BROWN] Brown, R.; Use of Analogy to Achieve New Expertise; MIT-AI Memo AI-TR-403, Massachusetts Institute of Technology Artificial Intelligence Laboratory; April 1977;

[CHANG] Chang, C. L.; Finding Missing Joins for Incomplete Queries in Relational Data Bases; Report RJ2145, IBM Research Laboratory, San Jose; February 1978;

[CODD] Codd, E. F., Arnold, R.S., Cadiou, J-M., Chang, C. L. and Roussopoulos, N.; RENDEZVOUS Version 1: An Experimental English-Language Query Formulation System for Casual Users of Relational Data Bases; Report RJ2144, IBM Research Laboratory, San Jose; January 1978;

[DAVIS] Davis, R.; Applications of Meta Level Knowledge to the Construction, Maintenance and Use of Large Knowledge Bases; Stanford Artificial Intelligence Laboratory Memo AIM-283; July 1976;

[FEIGENBA] Feigenbaum, E. A., Buchanan, B. G., and Lederberg, J.; On Generality and Problem Solving: A Case Study Using the DENDRAL Program; in Meltzer,B. and Michie, D. (eds.) Machine Intelligence 6, Endinburgh University Press, Endinburgh; 1971;

[FININ79] Finin, Timothy, Goodman, Bradley and Tennant, Harry; JETS: Achieving Completeness through Coverage and Closure; Proceedings of the 6th International Joint Conference on Artificial Intellegence; August 1979 (to appear);

[GINSPA78] Ginsparg, J. M.; Natural Language Processing in an Automatic Programming Domain; Stanford Artificial Intelligence Laboratory Memo AIM-316; June 1978;

[GOODMA77] Goodman, B. A.; A Model for a Natural Language Data Base System; Advanced Automation Group, Coordinated Science Laboratory, University of Illinois, Report R-798; October 1977;

[GREEN] Green, C., Gabriel, R., Ginsparg, J., Kant, E., Ludlow, J., Phillips, J., Steinberg, L., Tappel, S. and Westfold, S.; _Progress Report on Knowledge Based Programming_; System Control, Inc., Palo Alto; September 1978;

[HALLIDAY] Halliday, M. A. K.; Functional Diversity in Language as Seen from a Consideration of Modality and Mood in English; _Foundations of Language_, vol. 6, pp. 322-61; 1970;

[HAYES] Hayes, J. R., Waterman, D. A. and Robinson, C. S.; Identifying the Relevant Aspects of a Problem Text; _Cognitive Science_, Vol. 1, pp. 297-313; 1977;

[HEMPHILL] Hemphill, L. and Rhyne, J.; Models for Knowledge Bases in Natural Language Query Systems; (submitted for publication);

[MCDONALD] McDonald, D. and Hayes-Roth, F.; Inferential Searches of Knowledge Networks as an Approach to Extensible Language Understanding Systems; in Waterman and Hayes-Roth (eds.) _Pattern-Directed Inference Systems_, Academic Press; 1978;

[MCDERM77] McDermott, D.; Vocabularies for Problem Solver State Descriptions; _Proceedings of the 5th International Joint Conference on Artificial Intellegence_; August 1977;

[MINSKY75] Minsky, M.; A Framework for Representation Knowledge; in Winston (ed.) _The Psychology of Computer Vision_, McGraw Hill, New York; 1975;

[MOORE] Moore, J. and Newell, A.; How can Merlin Understand?; in _Knowledge and Cognition_, Gregg, L.(ed.), Lawrence Erlbaum Associates, Publishers, Potomac, Maryland; 1974;

[NALDA] NALDA (Naval Air Logistics Data Analysis) System Data Requirements Determination Report; Naval Aviation Integrated Support Center, Patuxent River, Maryland; 1975;

[NEWELL72] Newell, A. and Simon, H.; _Human Problem Solving_; Prentice-Hall, Inc., Englewood, New Jersey, pp. 72-75; 1972;

[POLYA57] Polya, G.; _How To Solve It_; Doubleday & Company, Inc., Garden City, New York; 1957;

[POLYA62] Polya, G.; _Mathematical Discovery_, vol. 1; John Wiley & Sons, Inc.; 1962;

[POPLE] Pople, H. E., Jr.; The Formation of Composite Hypothese in Diagnostic Problem Solving--An Exercise in Synthetic Reasoning; Proceedings of the 6th International Joint Conference on Artificial Intellegence; August 1977;

[RAPHAEL] Raphael, B.; THE THINKING COMPUTER Mind Inside Matter; W. H. Freeman and Company, San Francisco; 1976;

[ROBERTS] Roberts, B. R., and Goldstein, I. P.; The FRL Manual; MIT-AI Memo 409, Massachusetts Institute of Technology Artificial Intelligence Laboratory; September 1977;

[SHORTLIF] Shortliffe, E. H., et al.; An Artificial Intelligence Program to Advise Physicians Regarding AntiMicrobial Therapy; Computers and Biomedical Research, vol. 6, pp. 544-60; 1973;

[SOWA] Sowa, J. F.; Conceptual Graphs for a Data Base Interface; IBM Jour. of Research Developement; July 1976;

[TENNAN79] Tennant, Harry; Experience with the Evaluation of Natural Language Question Answerers; Proceedings of the 6th International Joint Conference on Artificial Intellegence; August 1979 (to appear);

[TENNAN80] Tennant, Harry; Natural Language Processing; Petrocelli, Inc., New York; (to be published in 1980);

[WALTZ76] Waltz, D.L., Finin, T.W., Green, F., Conrad, F., Goodman, B. and Hadden, G.; The PLANES system: Natural Language Access to a Large Data Base; Technical Report T-34, Coordinated Science Laboratory, University of Illinois; November 1976;

[WALTZ77] Waltz, D. L. and B. A. Goodman; Writing a Natural Language Data Base System; Proceedings of the 5th International Joint Conference on Artificial Intellegence; 1977;

[WALTZ78] Waltz, D. L.; An English Language Question Answering System for a Large Relational Data Base; CACM, vol. 21, pp. 526-539.; July, 1978;

[WALTZ79] Waltz, D. L.; Relating Images, Concepts and Words; Proceedings of the 6th International Joint Conference on Artificial Intellegence; August 1979 (to appear);

[WEBSTER] Webster's Dictionary; Fawcett Publications, Inc., Greenwich, Connecticut; 1974;

[WOODS72] Woods, W.A., Kaplan, R.M., Nash-Webber, B.; The Lunar Sciences Natural Language Information System: Final report; Bolt Beranek and Newman report No. 2378; June 1972;

[WOODS75] Woods, W.A.; Whats in a Link: Foundations for Semantic Networks; in D.G. Bobrow and A.M. Collins, eds,Representation and Understanding: Studies in Cognitive Science, Academic Press; 1975;

[WOODS77] Woods, W. A.; A Personal View of Natural Language Understanding; SIGART Newsletter, number 61, pp. 17-20; February 1977;

DATE
ILMED

-8